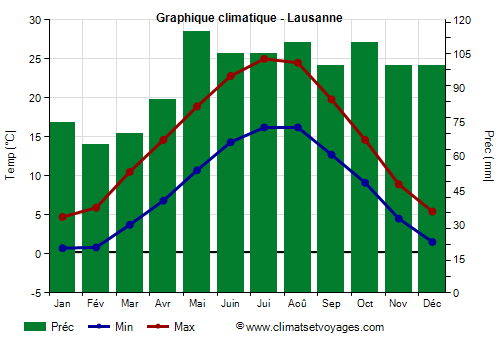
P\_FUN

Concevoir un logiciel pour afficher des graphiques sur des données



Matias Denis

ETML – FID2

Vennes – Lausanne

Table des matières

[Introduction du projet 3](#_Toc210902500)

[Objectifs du jeu 3](#_Toc210902501)

[Objectifs pédagogiques 3](#_Toc210902502)

[Domaine d’application 4](#_Toc210902503)

[Planification 4](#_Toc210902504)

[Production 5](#_Toc210902505)

[Organisation du code 5](#_Toc210902506)

[Choix techniques 5](#_Toc210902507)

[Problèmes rencontrés et solutions 5](#_Toc210902508)

[Qualité et bonnes pratiques 6](#_Toc210902509)

[Rapport de tests 7](#_Toc210902510)

[Journal de travail 7](#_Toc210902511)

[Usage de l’IA 7](#_Toc210902512)

[Conclusion 7](#_Toc210902513)

# Introduction du projet

Ce projet a pour but de réaliser un logiciel pour afficher des graphiques sur des données.

## Objectifs du jeu

Les objectifs du jeu sont :

* Réaliser un programme informatique de qualité.
  + Organisé
  + Compacté
  + Optimisé
  + Testé
  + Commenté
  + Complet
* Prouver que l’on est digne de confiance.
  + Journal de travail à jour
  + Pro-activité
    - Poser des questions au client
    - Faire des démonstrations
    - Utiliser un système de versioning de code (GIT)

## Objectifs pédagogiques

L’objectif principal de ce projet est de mettre en pratique les compétences acquises en C# et en gestion de projet à travers la réalisation d’une application complète de visualisation de séries temporelles.

Plus précisément, ce projet vise à :

* Gérer un mini-projet logiciel : définir des User Stories, planifier les tâches, tenir un journal de travail et livrer une version finale sur GitHub.
* Appliquer de bonnes pratiques de programmation : code structuré, utilisation de LINQ et d’extensions C#, gestion des erreurs et documentation.
* Traiter et importer des données réelles (CSV) de manière robuste.
* Concevoir une interface WPF fonctionnelle permettant d’afficher plusieurs séries temporelles, de filtrer par date et d’interagir avec le graphique (zoom, reset)
* Intégrer des fonctions mathématiques dans la visualisation pour enrichir l’analyse.
* Adopter une posture professionnelle : autonomie, communication avec le « client » et respect des consignes.

## Domaine d’application

Le domaine d’application dans lequel le logiciel est utilisé est la météo en Suisse. Plus précisément, les précipitations mesurées dans quatre villes suisses : Lausanne, Davos, Zürich et Lugano depuis environ 150 ans.

Ces données ont été récupérées sur le site officiel de météorologie et de climatologie MétéoSuisse.

# Planification

La planification a été faite sur GitHub Projects sous forme de monthly roadmap.

* 25 août 2025 -> Prise en main du projet
* 1 septembre 2025 -> Création de la maquette de l’interface graphique du produit.
* 8 septembre 2025 -> Début du code et donc de l’importation des fichiers CSV.
* 15 septembre 2025 -> Affichage d’une courbe simple.
* 29 septembre 2025 -> Affichage de plusieurs courbes, séries de données importées.
* 6 octobre 2025 -> Implémentation de la navigation et du zoom.
* 27 octobre 2025 -> Implémentation d’un mode de fonctions mathématiques.

La planification finale n’a pas été respectée car il y a eu des cas de maladie et d’accident, ce qui a apporté du retard. De plus, les tâches à faire ont été d’un niveau de complexité supérieur au niveau attendu.

# Production

Le projet est construit en C# avec Windows Forms pour l’interface graphique et ScottPlot v5 pour l’affichage des graphiques.

L’application permet de :

* Importer des données météorologiques (précipitations) depuis un fichier CSV (format important)
* Visualiser ces données sous formes de courbes
* Filtrer par ville et par période
* Appliquer éventuellement une fonction mathématique de démonstration

## Organisation du code

**MainWindow.xaml.cs** : contient la logique de l’interface. On y trouve les gestionnaires d’événements (boutons, cases à cocher/décocher, sélection de dates) et la méthode **Redraw()** qui met à jour le graphique.

**MeteoSuisseCsvImporter.cs :** classe utilitaire qui lit les fichiers CSV, détecte le séparateur, gère l’encodage, et transforme chaque ligne en objet **TimePoint**.

**TimePoint**(record) :représente un couple **(Date, Valeur)** correspondant à une mesure de précipitations.

**Extensions LINQ**: ajoutées pour faciliter le filtrage des données par période, obtenir les bornes, etc…

**FunctionFactory**: permet d’ajouter des fonctions mathématiques à tracer (utilisé comme démonstration) de fonctionnalités de ScottPlot).

## Choix techniques

ScottPlot v5 a été choisi pour la simplicité de génération de graphiques interactifs.

Le code utilise LINQ au lieu de boucles for, ce qui rend le traitement des séries plus lisible et concis.

La lecture du CSV a été rendue robuste : détection automatique du séparateur ( ; ou , ou tabulation), prise en compte de différents formats de date (ex : dd.mm.yyyym yyyy-mm-dd), gestion des décimales avec virgules ou points.

Une attention particulière a été portée à l’encodage (UTF-8 / Latin-1), car les fichiers contiennent des accents qui causent des erreurs si mal décodées.

## Problèmes rencontrés et solutions

**Encodage**: les fichiers de MétéoSuisse étaient en Latin-1, ce qui empêchait la reconnaissance correcte des colonnes. La solution a été de forcer la lecture avec **Encoding.Latin1**.

**Variation des en-tête CSV**: certains fichiers n’avaient pas exactement les mêmes intitulés de colonnes. La solution a été de normaliser les nom (minuscule, sans accents).

## Qualité et bonnes pratiques

Les méthodes sont courtes et bien séparées (import, affichage, extensions LINQ)

Le code est largement commenté et respecte les conventions de nommage de l’ETML (identifiants explicites, variables en camelCase, classes en PascalCase).

Les exceptions sont gérées proprement dans l’import (erreurs de format, dates invalides, séparateur inconnu). L’utilisateur reçoit un message clair dans la zone de statut.

# Rapport de tests

Scénario 1 — US1 : Navigation et zoom

Objectif : Vérifier le zoom et le déplacement du graphique, et le bouton Réinitialiser.

Arrange / Given

* Lancer l’application PTL avec au moins une série affichée (ex : Lausanne).
* Vue initiale = affichage global (toutes les années disponibles).

Act / When

1. Utiliser la molette de la souris sur la zone du graphique (zoom avant/arrière).
2. Cliquer-glisser (bouton gauche) sur la zone du graphique pour panner (déplacer la vue).
3. Cliquer sur le bouton Réinitialiser.

Assert / Then

* Zoom avec la molette modifie l’échelle visible sur l’axe X et/ou Y.
* Le clic-glisser permet de déplacer la fenêtre visible (pan).
* Après clic sur Réinitialiser, l’affichage ne revient pas exactement à la vue globale initiale (min/max dates et échelle d’origine).

Résultat : KO

Remarque : Tester sur plusieurs niveaux de zoom.

Scénario 2 — US2 : Afficher plusieurs séries

Objectif : Permettre la comparaison : importer et afficher plusieurs séries simultanément.

Arrange / Given

* Préparer 2 fichiers CSV distincts (ex : Lausanne.csv, Zürich.csv) ou 2 fichiers couvrant des années différentes.
* Application ouverte, pas de séries encore importées.

Act / When

1. Cliquer sur Importer un fichier → sélectionner le premier CSV → valider.
2. Répéter pour le second CSV.
3. Vérifier que les deux séries sont cochées/activées dans la liste des séries.

Assert / Then

* Les deux séries apparaissent simultanément sur le graphique.
* Chaque série a une couleur différente (visuellement distincte).
* La légende liste les deux séries avec un identifiant clair (ex : "Lausanne", "Zürich") et on peut repérer quelle couleur correspond à quelle série.

Résultat : OK

Remarque : Tester aussi l’import de 3+ fichiers, vérifier gestion des noms identiques (conflit) et comportement de la légende quand il y a de nombreuses séries.

Scénario 3 — US3 : Afficher une courbe simple

Objectif : Vérifier l’affichage d’une série simple (températures dans le temps).

Arrange / Given

* Avoir un fichier CSV valide contenant colonnes Date et heure et Précipitations (mm) ou Température (adapter le mapping si nécessaire).
* Application ouverte.

Act / When

1. Importer le fichier CSV (bouton Importer un fichier).
2. S’assurer que la série importée est cochée/affichée.

Assert / Then

* Axe X affiche des dates (graduations lisibles, ex : ticks annuels).
* Axe Y affiche la valeur (températures / précipitations) avec unité si possible.
* La légende affiche le nom de la série (ex : Lausanne 2024) et on le voit clairement sur l’UI.

Résultat : OK

Remarque : Si le CSV contient des valeurs par année (01.01.YYYY) la fenêtre doit les interpréter comme dates annuelles. Vérifier l’échelle lorsque les dates sont en format dd.MM.yyyy HH:mm.

Scénario 4 — US4 : Importation de fichiers CSV

Objectif : Valider l’import CSV (date + valeur) et le traitement en objets TimePoint.

Arrange / Given

* Avoir un bouton Importer un fichier visible.
* Préparer : 1 CSV valide (format attendu : Ville;Date et heure;Précipitations (mm);Jours de précipitations) et 1 CSV invalide (ex : colonne manquante).

Act / When

1. Cliquer sur Importer un fichier et choisir le CSV valide.
2. Observer message de statut / comportement.
3. Répéter avec le CSV invalide.

Assert / Then

* Pour le fichier valide : le parser crée une liste d’objets (Date, Valeur) et la série est ajoutée dans \_series.
* Un message de confirmation s’affiche : *“Fichier chargé avec succès !”* ou équivalent.
* Pour le fichier invalide : afficher un message d’erreur clair (ex : *“Fichier invalide. Vérifiez les colonnes et le séparateur.”*), et aucune exception non gérée ne doit planter l’application.

Résultat : OK

Remarque : Tester valeurs manquantes et doublons (agrégation par date attendue).

Scénario 5 — US5 : Mode fonctions mathématiques

Objectif : Permettre le tracé de fonctions (prédéfinies + personnalisée).

Arrange / Given

* L’application ouverte, onglet/contrôle Fonctions accessible (ComboBox / onglet).
* Champ texte disponible (pour saisie fonction personnalisée).

Act / When

1. Sélectionner une fonction prédéfinie (ex : x^2) dans la liste.
2. Visualiser le graphe.
3. Saisir une expression personnalisée (ex : sin(x) + 0.5\*x) dans le champ texte et valider (bouton *Tracer* ou *Entrée*).
4. Tester une expression invalide (ex : sin() ou abc) et valider.

Assert / Then

* La sélection d’un preset trace correctement la courbe correspondante.
* L’expression personnalisée est évaluée (via Roslyn ou parser) et la courbe s’affiche ; en cas d’erreur, afficher un message d’erreur lisible (ex : *“Expression invalide : erreur de compilation”*).
* Les fonctions utilisent un domaine visuel par défaut (ex : x ∈ [-10,10]) avec possibilité d’ajuster (idéalement via champs min/max).

Résultat : KO

Remarque : Tester expressions utilisant sin, cos, ^ (mapped to Math.Pow) et nombres décimaux.

# Journal de travail

Le journal de travail retraçant toutes les étapes des différentes séquences est disponible dans le GitHub du projet. Le lien pour s’y rendre est ici https://github.com/Jazztel20/P\_FUN/blob/main/JDT/T-P\_FUN-MatiasDenis-JDT.xlsx

# Usage de l’IA

L’IA a servi de guide dans ce projet car les bases, jusqu’ici acquises en C#, ne permettent pas de produire un projet comme celui-ci sans aide externe. De plus, tous les projets antérieurs ont été effectués en console et non en Windows Forms, ce qui rajoute de la difficulté. Il a été demandé à l’IA de produire ce qui prenait le plus de temps, le code, le debug des erreurs, ainsi que les rapports de tests fonctionnels.

# Conclusion

Ce projet est très difficile à faire pour une personne n’ayant pas le niveau requis en C# et en Windows Forms. Beaucoup de fonctionnalités comme l’import de CSV, l’affichage d’une fenêtre graphique où les données des fichiers CSV sont affichées sont des concepts nullement vus en cours. Le code basique et, lui aussi, dur à produire car les bases de C# ne sont pas acquises de mon côté. Je trouve ce projet peu pertinent aux vues du niveau global en programmation C# & Windows Forms. Je n’y ai trouvé que très peu d’intérêt.